Robust Approximate Aggregation in Sensor Data Management Systems

JEFFREY CONSIDINE
Boston University
MARIOS HADJIELEFTHERIOU
AT&T Labs
and
FEIFEI LI, JOHN BYERS, and GEORGE KOLLIOS
Boston University

In the emerging area of sensor-based systems, a significant challenge is to develop scalable, fault-tolerant methods to extract useful information from the data the sensors collect. An approach to this data management problem is the use of sensor database systems, which allow users to perform aggregation queries such as MIN, COUNT, and AVG on the readings of a sensor network. In addition, more advanced queries such as frequency counting and quantile estimation can be supported. Due to energy limitations in sensor-based networks, centralized data collection is generally impractical, so most systems use in-network aggregation to reduce network traffic. However, even these aggregation strategies remain bandwidth-intensive when combined with the fault-tolerant, multipath routing methods often used in these environments. To avoid this expense, we investigate the use of approximate in-network aggregation using small sketches. We present duplicate-insensitive sketching techniques that can be implemented efficiently on small sensor devices with limited hardware support and we analyze both their performance and accuracy. Finally, we present an experimental evaluation that validates the effectiveness of our methods.

Categories and Subject Descriptors: H.2.m [**Database Management**]: Miscellaneous

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Sensor databases, approximation algorithms, aggregation, sketches, synopses

6

J. Byers and J. Considine are supported in part by NSF grants ANI-9986397, ANI-0093296, and ANI-0205294. G. Kollios and F. Li are supported in part by NSF grants IIS-0133825 and IIS-0308213.

Authors' addresses: J. Considine, Department of Computer Science, Boston University, 111 Cummington Street, Boston, MA 02215; M. Hadjieleftheriou, AT&T Labs, Inc., Florham Park, NJ; F. Li, J. Byers, and G. Kollios (corresponding author), Department of Computer Science, Boston University, 111 Cummington Street, Boston, MA 02215; email: gkollios@cs.bu.edu.

DOI 10.1145/1508857.1508863 http://doi.acm.org/10.1145/1508857.1508863

ACM Reference Format:
Considine, J., Hadjieleftheriou, M., Li, F., Byers, J., and Kollios, G. 2009. Robust approximate aggregation in sensor data management systems. ACM Trans. Database Syst. 34, 1, Article 6 (April 2009), 35 pages. DOI = 10.1145/1508857.1508863 http://doi.acm.org/ 10.1145/1508857.1508863

1. INTRODUCTION

As computation-enabled devices shrink in scale and proliferate in quantity, a relatively recent research direction has emerged to contemplate future applications of these devices and services to support them. A canonical example of such a device is a *sensor mote*, a device with measurement, communication, and computation capabilities, powered by a small battery [Horton et al. 2002]. Individually, these motes have limited capabilities, but when a large number of them are networked together into a *sensor network*, they become much more capable. Indeed, large-scale sensor networks are now being applied experimentally in a wide variety of areas; some sample applications include environmental monitoring, surveillance, and traffic monitoring.

In a typical sensor network, each sensor produces a stream of sensory observations across one or more sensing modalities, for example, one for sensing temperature, another for sensing humidity, a third sensing acceleration, etc. But for many applications and sensing modalities, it is unnecessary for each sensor to report its entire data stream in full fidelity. Moreover, in a resource-constrained sensor network environment, each message transmission is a significant, energy-expending operation. For this reason, and because individual readings may be noisy or unavailable, it is natural to use data aggregation to summarize information collected by sensors. As a reflection of this, a database approach to managing data collected on sensor networks has been advocated [Yao and Gehrke 2002; Madden et al. 2005], with particular attention paid to efficient query processing for aggregate queries [Madden et al. 2002; Yao and Gehrke 2002; Zhao et al. 2003].

In the TinyDB system [Madden et al. 2005, 2003, 2002], users connect to the sensor network using a workstation or base station directly connected to a sensor designated as the sink. Aggregate queries over the sensor data are formulated using a simple SQL-like language, then distributed across the network. Aggregate results are sent back to the workstation over a spanning tree, with each sensor combining its own data with results received from its children. If there are no failures, this in-network aggregation technique is both effective and energy efficient for distributive and algebraic aggregates [Gray et al. 1997] such as MIN, MAX, COUNT, and AVG. However, as we will argue, this technique is much less effective in sensor network scenarios with moderate node and link failure rates. Node failure is inevitable when inexpensive, faulty components are placed in a variety of uncontrolled or even hostile environments. Similarly, link failures and packet losses are common across wireless channels because of environmental interference, packet collisions, and low signal-to-noise ratios [Zhao et al. 2003].

When a spanning tree approach is used for aggregate queries, a single failure results in an entire subtree of values being lost. If this failure is close to the

sink, the change in the resulting aggregate can be significant. Retransmission-based approaches are expensive in this environment, so solutions based upon multipath routing have been proposed [Madden et al. 2002]. For aggregates such as MIN and MAX that are monotonic and exemplary, this provides a fault-tolerant solution. But for duplicate-sensitive aggregates such as COUNT or AVG that give incorrect results when the same value is counted multiple times, existing methods are not satisfactory.

In this article, we propose a robust and scalable method for computing duplicate-sensitive aggregates across faulty sensor networks. Guaranteeing exact solutions in the face of losses is generally impractical, so we instead consider approximate methods. These methods are robust against both link and node failures. Our contributions can be summarized as follows.

—We present a method to combine duplicate-insensitive sketches with multipath in-network aggregation techniques to produce accurate approximations with low communication and computation overhead.

—We extend well-known duplicate-insensitive sketches [Flajolet and Martin 1985] to handle SUM aggregates. We present a new algorithm that improves the insertion time of a new item c_i into the sketch from $O(\log^2 c_i)$ to $O(\log c_i)$. Furthermore, our algorithm can work on small sensor devices with low-performance hardware.

—We discuss how to extend existing sketching techniques for frequency estimation and quantile computation and make them duplicate-insensitive and compatible with multipath routing protocols. In particular, we discuss and analyze methods to extend the well-known Count-Min [Cormode and Muthukrishnan 2005a] and Quantile Digest [Shrivastava et al. 2004] sketches.

—Finally, we present an extensive experimental evaluation of our proposed methods in comparison with previous approaches.

The remainder of this article proceeds as follows. Background material is covered in Section 2. A new efficient algorithm to handle SUM aggregates with duplicate-insensitive sketches is discussed in Section 3. Frequency counting sketches are presented in Section 4. Quantile estimation sketches are presented in Section 5. We validate our methods experimentally in Section 6 and we provide a discussion on the experimental results in Section 7. Finally, we conclude in Section 8.

2. BACKGROUND

We now briefly survey related work. Sensors and their limitations are described in Section 2.1. Previous frameworks for processing aggregates are covered in 2.2, and multipath routing techniques are covered in 2.3. Finally, the sketches which we use to improve upon these frameworks are introduced in Section 2.4.

2.1 Sensor Devices

Today's sensor motes (e.g., Horton et al. [2002]) are full-fledged computer systems, with a CPU, main memory, operating system, and a suite of sensors. They

are powered by small batteries and their lifetime is primarily dependent on the extent to which battery power is conserved. The power consumption tends to be dominated by transmitting and receiving messages and most systems try to minimize the number of messages in order to save power. Also, the communication between sensors is wireless and the packet loss rate between nodes can be high. For example, Zhao et al. [2003] reports on experiments in which more than 10% of the links suffered average loss rate greater than 50%. Another challenge is that links may be asymmetric, both in loss rates and even reachability. These limitations motivate the design of new query evaluation methods that are more appropriate for sensor network environments. First, the query execution plan must be energy efficient and second, the process must be as robust as possible, given the communication limitations in these networks. Finally, we assume that each sensor node is assigned a unique id, which is used in our methods.

2.2 In-Network Aggregate Query Processing

A simple approach to evaluate an aggregation query is to route all sensed values to the base station and compute the aggregate there. Although this approach is simple, the number of messages and the power consumption can be large. A better approach is to leverage the computational power of the sensor devices and compute aggregates in-network. Aggregates that can be computed in-network include all decomposable functions [Madden et al. 2002].

Definition 1. A function f is decomposable if it can be computed by another function g as follows: $f(v_1, v_2, ..., v_n) = g(f(v_1, ..., v_k), f(v_{k+1}, ..., v_n))$.

Using decomposable functions, the value of the aggregate function can be computed for disjoint subsets, and these values can be used to compute the aggregate of the whole using the merging function g. Our discussion is based on the TinyDB [Madden et al. 2005] system. However, similar approaches are used to compute aggregates in other systems [Yao and Gehrke 2002, 2003, Zhao et al. 2003; Intanagonwiwat et al. 2003]. TinyDB is a query processing system that uses an SQL-like language to extract information from a sensor network. In particular, TinyDB uses a database view of the sensor network. Sensor measurements belong to a single (logical) table called *sensors*, which has one tuple for each node per time instant. The attributes of this table include the sensor id, a temporal attribute, and one attribute for each sensing modality. In addition, TinyDB allows to store other tables called *materialized points*, which are similar to materialized views in relational databases. An example of an aggregation query in TinyDB is the following.

```
SELECT COUNT(*)
FROM sensors
WHERE temperature >  30
SAMPLE Period 1s FOR 10s
```

This query computes the number of sensors that sense a temperature value more than 30° Celsius. The query is executed every 1 second for 10 seconds.

Therefore, the result will be 10 values for 10 different time instants. Details about the query capabilities of TinyDB are found in Madden et al. [2005].

TinyDB executes the queries using power-efficient methods and in-network query processing. To perform aggregation queries efficiently with low power consumption, it uses the Tiny AGgregation (TAG) framework [Madden et al. 2002]. In TAG, the in-network query evaluation has two phases, the *distribution* phase and the *collection* phase. During the distribution phase, the query is flooded in the network and organizes the nodes into an *aggregation tree*. The base station broadcasting the query is the *root* of the tree. The query message has a counter that is incremented with each forwarding and counts the hop distance from the root. In this way, each node is assigned to a specific level equal to the node's hop distance from the root. Also, each sensor chooses one of its neighbors with a smaller hop distance from the root to be its parent in the aggregation tree.

During the collection phase, each leaf node produces a single tuple and forwards this tuple to its parent. The nonleaf nodes receive the tuples of their children and combine these values. Then, they submit the new partial results to their own parents. This process runs continuously and after h steps, where h is the height of the aggregation tree, the aggregate result arrives at the root. In order to conserve energy, sensor nodes sleep as much as possible, that is, whenever the processor and radio are idle. When a timer expires or an external event occurs, the device wakes and starts the processing and communication phases. At this point, it receives the messages from its children and then submits the new value(s) to its parent. After this, if no more processing is needed for that step, it enters again into the sleeping mode [Madden et al. 2003].

As mentioned earlier, this approach works very well for ideal network conditions, but is less effective under lossy conditions. To address this problem, TAG uses a number of different methods [Madden et al. 2002]. One solution is to cache previous values and reuse them if newer ones are unavailable. Of course, these cached values may reflect losses at lower levels of the tree. Another approach considered in this same work takes advantage of the fact that a node may select multiple parents from neighbors at a higher level. Using this approach, which we refer to as "fractional parents," the aggregate value is decomposed into fractions equal to the number of parents. Each fraction is then sent to a distinct parent instead of sending the whole value to a single parent. For example, given an aggregate sum of 15 and 2 parents, each parent would be sent the value 7.5. It is easy to demonstrate analytically that this approach does not improve the expected value of the estimate over the single-parent approach; it only helps to reduce the variance of the estimated value at the root. Therefore, the problem of losing a significant fraction of the aggregate value due to network failures remains. Some recent work has addressed the aforementioned problem in sensor networks [Considine et al. 2004; Nath et al. 2004; Manjhi et al. 2005; Kamra et al. 2007]. In addition, similar techniques can be used for aggregation estimation in peer-to-peer networks [Kempe et al. 2003; Bawa et al. 2003]. In this article, we propose new methods to address more general aggregation functions with better complexity bounds.

There has been much additional work on sensor network data management systems in the past few years, notably on sensor data modeling and data compression [Deshpande et al. 2004; Deshpande and Madden 2006; Kotidis 2005; Deligiannakis et al. 2004a]. In an effort to minimize the number of messages and the bandwidth consumed by a TAG-like system, other recent methods use past values and provide an error guarantee on the aggregation result [Deligiannakis et al. 2007, 2004b, Sharaf et al. 2004]. These techniques are orthogonal to our methods and may be combined with the techniques proposed in this article. Finally, a promising new direction in sensor networks is to consider security and authentication problems. Some recent papers [Przydatek et al. 2003; Garofalakis et al. 2007] address issues in this area using as a base technique the sketches that we also use in this article.

2.3 Best-Effort Routing in Sensor Networks

Recent years have seen significant work on best-effort routing in sensor and other wireless networks. Due to high loss rates and power constraints, a common approach is to use multipath routing, where more than one copy of a packet is sent to the destination over different paths. For example, directed diffusion [Intanagonwiwat et al. 2003] uses a flood to discover short paths that sensors use to send back responses. Various positive and negative reinforcement mechanisms are used to improve path quality. Braided diffusion [Ganesan et al. 2001] builds on directed diffusion to use a set of intertwined paths for increased resilience. A somewhat different approach is used by GRAB [Ye et al. 2005], where paths are not explicitly chosen, but the width of the upstream broadcast is controlled.

2.4 Distinct Counting Sketches

Counting sketches (FM sketches) were introduced by Flajolet and Martin [1985] for the purpose of quickly estimating the number of distinct items in a database (or stream) in one pass while using only a small amount of space. Since then, there has been much work developing and generalizing counting sketches (e.g., Alon et al. [1996], Bar-Yossef et al. [2002], Cormode et al. [2003], Flajolet [1990], Ganguly et al. [2003], Gibbons and Tirthapura [2001]).

It is well known that an exact solution to the distinct counting problem over a multiset with n items requires $\Omega(n)$ space. However, only $\Theta(\log n)$ space is required to *approximate* the number of distinct items [Alon et al. 1996]. The original FM sketches achieve this bound [Flajolet and Martin 1985], though they assume a fixed hash function that appears random, so they are vulnerable to adversarial choices of inputs. We use these sketches since they are very small and accurate in practice, and describe them in detail next. We also mention that a different sketching scheme using linear hash functions has been proposed, that can address adversarial input [Alon et al. 1996]. These sketches are somewhat larger than FM sketches in practice, although a recent technique [Durand and Flajolet 2003] extends these methods and uses only $O(\log \log n)$ space.

First, we define the distinct counting problem, and present details of the FM sketches of Flajolet and Martin [1985] that provide the key property of

duplicate insensitivity that we exploit along with necessary parts of the theory behind them.

Definition 2. Given a multiset of items $M = \{x_1, x_2, x_3, \dots\}$, the *distinct counting* problem is to compute $n \equiv \left|\text{distinct}(M)\right|$.

Given a multiset M, the FM sketch of M, denoted $S(M)$, is a bitmap of length k. The entries of $S(M)$, denoted $S(M)[0, \dots, k-1]$, are initialized to zero and are set to one using a random binary hash function h applied to the elements of M. Formally,

$$S(M)[i] \equiv 1 \text{ iff } \exists x \in M \text{ s.t. } \min\{j \mid h(x, j) = 1\} = i.$$

By this definition, each item x sets a single bit in $S(M)$ to one; the bit indexed by the minimum i for which $h(x, i) = 1$. This gives a simple serial implementation which is very fast in practice, as shown in Algorithm 1. The number of times needed to call the hash function for an item follows a geometric distribution with $p = \frac{1}{2}$, since every call of the hash function is a Bernoulli trial with a fixed value of p [Papoulis 1965]. The expected value of the geometric distribution is $\frac{1}{p} = 2$. Thus, we have the following claim.

Claim 1. An item x_i can be inserted into an FM sketch in $O(1)$ expected time.

Algorithm 1. CountInsert(S,x)

```
1: i = 0;
2: while hash(x,i) = 0 do
3: i = i + 1;
4: end while
5: S[i] = 1;
```

We also employ the following useful properties of FM sketches noted in Flajolet and Martin [1985].

Property 1. *The FM sketch of the union of two multisets is the bit-wise OR of their FM sketches; that is,* $S(M_1 \cup M_2)[i] = (S(M_1)[i] \vee S(M_2)[i])$.

Property 2. $S(M)$ *is entirely determined by the distinct items of* M. *Duplication and insertion order do not affect* $S(M)$.

Property 1 allows each node to compute a sketch of locally held items and send the small sketch for aggregation elsewhere. Since aggregation via union operations is inexpensive, it may be performed in the network without significant computational burden. Property 2 allows the use of multipath routing of the sketches for robustness without affecting the accuracy of the estimates.

The next lemma provides key insight into the behavior of FM sketches and will be the basis of efficient implementations of summation sketches later.

Lemma 1. *For* $i < \log_2 n - 2\log_2\log_2 n$, $S(M)[i] = 1$ *with probability* $1 - O(ne^{-\log_2^2 n})$. *For* $i \geq \frac{3}{2}\log_2 n + \delta$, *with* $\delta \geq 0$, $S(M)[i] = 0$ *with probability* $1 - O\left(\frac{2^{-\delta}}{\sqrt{n}}\right)$.

PROOF. This lemma is proven in Flajolet and Martin [1985] and follows from basic balls-and-bins arguments. □

The lemma implies that, given an FM sketch of n distinct items, we expect an initial prefix of all ones and a suffix of all zeros, while only the setting of the bits around $S(M)[\log_2 n]$ exhibit much variation. This gives a bound on the number of bits k required for $S(M)$ in general: $k = \frac{3}{2}\log_2 n$ bits suffice to represent $S(M)$ with high probability. It also suggests that just considering the *length* of the prefix of all ones in this sketch can produce an estimate of n. Formally, let $R_n \equiv \min\{i \mid S(M)[i] = 0\}$ when $S(M)$ is an FM sketch of n distinct items. In other words, R_n is a random variable marking the location of the first zero in $S(M)$. In Flajolet and Martin [1985], a method to use R_n as an estimator for n is developed using the following theorems.

THEOREM 1. *The expected value of R_n for FM sketches satisfies $\mathbf{E}[R_n] = \log_2(\varphi n) + P(\log_2 n) + o(1)$, where the constant φ is approximately 0.775351 and $P(u)$ is a periodic and continuous function of u with period 1 and amplitude bounded by 10^{-5}.*

THEOREM 2. *The variance of R_n for FM sketches, denoted σ_n^2, satisfies $\sigma_n^2 = \sigma_\infty^2 + Q(\log_2 n) + o(1)$, where constant σ_∞^2 is approximately 1.12127 and $Q(u)$ is a periodic function with mean value 0 and period 1.*

Thus, R_n can be used for an unbiased estimator of $\log_2 \varphi n$ if the small periodic term $P(\log_2 n)$ is ignored. However, an important concern is that the variance is slightly more than one, dwarfing $P(\log_2 n)$, and implying that estimates of n will often be off by a factor of two or more in either direction.

To improve the variance and confidence of the estimator, FM sketches can use multiple bitmaps. Specifically, each item is inserted into each of m independent bitmaps to produce m values, $R^{\langle 1\rangle}, \ldots, R^{\langle m\rangle}$. The estimate is then calculated as follows.

$$n \approx (1/\varphi)2^{\sum_i R^{\langle i\rangle}/m}$$

This estimate is more accurate, with standard error $O(1/\sqrt{m})$, but comes at the cost of $O(m)$ expected insertion times. To avoid this overhead, an algorithm called Probabilistic Counting with Stochastic Averaging, or PCSA, was proposed in Flajolet and Martin [1985]. Instead of inserting each item into each of the m bitmaps, each item is hashed and inserted into only one. Thus, each of the bitmaps summarizes approximately n/m items. While there is some variation in how many items are assigned to each bitmap, further analysis showed that the standard error of PCSA is roughly $0.78/\sqrt{m}$. Using PCSA, insertion takes $O(1)$ expected time.

We discuss now how to answer the example COUNT query that we give in Section 2.2 using FM sketches with PCSA. Every sensor satisfying the WHERE predicate (e.g., has a temperature reading more than 30° Celsius) creates an FM sketch using its sensor id as a unique item. Therefore, every node creates an FM sketch with a single bit set to 1. Then, the protocol discussed in Section 2.3 is used to send the sketches up to the next level, until they reach the sink. Multiple downstream sketches are merged into a single upstream sketch using

bit-wise OR operations based on Properties 1 and 2. Finally, the sink node, upon receiving the final sketch of the query, uses the estimation function to compute an approximate value for the COUNT aggregate.

2.5 Sketch Sizes and Compression

As mentioned earlier, the other main limitation of sensor networks is their limited bandwidth. This limitation is a cause for concern when comparing sketching-based techniques against the spanning tree strategies of TAG. While 2 bytes of data per packet will generally suffice for TAG, a single sketch in our methods also uses 2 bytes and our later experiments will actually use 20 sketches per packet for a single aggregate. However, as implied in part by Lemma 1, these sketches are compressible. To leverage this, our experiments will use the compression techniques proposed by Palmer et al. [2002]. In brief, we use an adapted run-length encoding algorithm to compress the bitmaps in a given sketch. This reduces the space requirements to about 50% to 70% of the uncompressed versions. This is sufficient for two aggregates to be sketched within one TinyDB packet (up to 48 bytes).

2.6 Sketch-Based Aggregate Query Processing

All of the best-effort techniques discussed in Section 2.1 must address the problem of duplicated data. While it is relatively easy to detect duplicate *packets* in the network stack, it is much more difficult to detect data that has been duplicated within aggregates that may partially, but not completely, overlap. For example, a data value A might be aggregated with B along one braid of a diffusion, and with C along a different braid. Naively summing the two braids double-counts A, while explicitly deduplicating A would necessitate complex specification of packet contents. A better approach, that we advocate, is to use a duplicate-insensitive representation of the data, which admits much simpler multipath in-network aggregation as suggested in TAG. By utilizing duplicate-insensitive sketches we can build a robust, loss-resilient framework for aggregation. Our methods for aggregation leverage two main observations. First, the wireless communication of sensor networks gives the ability to broadcast a single message to multiple neighbors simultaneously. Second, the duplicate-insensitive sketches allow a sensor to combine all of its received sketches into a single sent message. Given proper synchronization, this allows us to aggregate data robustly, with each sensor sending just one broadcast.

Given a continuous aggregation query, the computation proceeds in two phases. In the first phase, the query is distributed across the sensor network, often using some form of flooding. During this phase, each node also computes its level (i.e., its hop distance from the root), and notes the level values of its immediate neighbors. The second phase is divided into a series of *epochs* specified by the query. The specified aggregate will be computed once for each epoch.

At the beginning of each epoch, each node constructs a sketch of its local values for the aggregate. The epoch is then subdivided into a series of rounds, one for each level, starting with the highest (farthest) level. In each round, the nodes at the corresponding level broadcast their sketches, and the nodes at

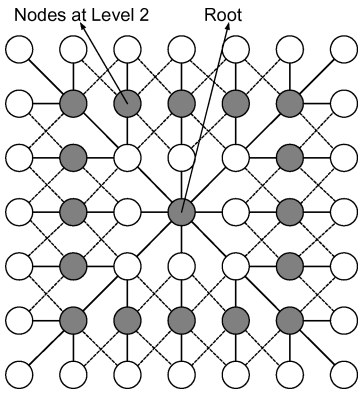


Fig. 1. Routing topology for 49-node grid.

the next level receive these sketches and combine them with their sketches in progress. In the last round, the sink receives the sketches of its neighbors, and combines them to produce the final aggregate.

As an example, we step through a single epoch aggregating over the topology of Figure 1. First, each node creates a fresh sketch summarizing its own observed values. In the first round of communication, nodes at level 3 broadcast their sketches, which are then received by neighboring level-2 nodes and combined with the sketches of the level-2 nodes. In the second round, nodes at level-2 broadcast their sketches, which are then received by neighboring level-1 nodes and combined with the sketches of the level-1 nodes. In the third and last round, nodes at level-1 send their sketches to the sink, which combines them and extracts the final aggregate value. Note that each node in this topology except those on the diagonals has multiple shortest paths which are effectively used, and a value will be included in the final aggregate unless all of its paths suffer losses.

The tight synchronization described so far is not actually necessary. Our methods can also be applied using gossip-style communication; the main advantage of synchronization and rounds is that better scheduling is possible and power consumption can be reduced. However, if a node receives no acknowledgments of its broadcast, it may be reasonable in practice to retransmit. More generally, loosening the synchronization increases the robustness of the final aggregate as paths taking more hops are used to route around failures. This increased robustness comes at the cost of power consumption, since nodes broadcast and receive more often (due to values arriving later than expected) and since computing the final aggregate takes longer.

3. SUMMATION SKETCHES

In this section we discuss robust and efficient methods to answer SUM aggregation queries in sensor networks. We assume that we have a set of sensors, where each sensor i stores a single value c_i. Our goal is to estimate the sum of these values $\sum_{i=1}^{N} c_i$. Again, we would like to use sketches and multipath in-network aggregation. As before, the value reported by a given sensor may

appear multiple times in the sink or any other intermediate sensor. To distinguish between a duplicate value of the same sensor and the same value coincidentally reported by a different sensor, we associate the sensor id with the value that sensor reports. Thus, we create pairs $\langle k_i, c_i \rangle$, where k_i is the unique sensor id and c_i is the value stored in this sensor. Therefore, to solve the SUM aggregation query using the in-network multipath aggregation technique, we need to solve the following problem first.

Definition 3. Let M be a multiset of pairs $\{\langle k_1, c_1 \rangle, \langle k_2, c_2 \rangle, \dots\}$ where k_i is a key (i.e., the sensor id for each sensor) and c_i is a non-negative integer value. The distinct summation problem is to calculate the sum of c_i for all distinct k_i.

$$DS \equiv \sum_{\text{distinct}(k_i \in M)} c_i$$

For summations involving small c_i values, we can estimate the distinct sum by using an FM sketch S. The idea is that, if we need to insert a new value c_i into S, we can deterministically create c_i unique items associated with k_i and insert them into the sketch one by one. The FM sketch estimator will thus be increased by c_i in expectation. Concretely, associate with k_i the distinct insertions $\langle k_i, c_i, 1 \rangle, \dots, \langle k_i, c_i, c_i \rangle$. By using purely deterministic routines for generating these per-pair insertions (e.g., by using a hash function seeded on k_i and c_i), duplicate pair insertions will have no effect on the FM sketch. Since this procedure performs c_i invocations of the FM insertion routine per key, the analysis for FM sketches applies; insertion takes $O(c_i)$ expected time. For large values of c_i this approach is not practical, leading us to consider more scalable alternatives.

Our basic premise is to efficiently configure the bits in the summation FM sketch *as if* we had performed c_i successive insertions, but without actually performing them. In other words, we try to efficiently emulate the outcome of c_i sequential insertions into an FM sketch, by drawing uniformly and reproducibly from the *same distribution* that these insertions would yield.

3.1 Summation Sketch Algorithm

The key observation behind our faster insertion algorithm is that in the course of updating the FM sketch with the insertion of key k_i, most of the c_i subinsertions attempt to set the same early bits; repeated attempts at setting the same bits accomplish nothing. Faster insertions for summation sketches are possible by skipping over ineffective repeat attempts.

We begin with some definitions. Let S be an FM sketch with a bitmap of length l, where all bits are initialized to zero. We say that an insertion $y = \langle k_i, c_i, x \rangle$ *reaches* bit z of the FM sketch if and only if the insertion CountInsert(S, y) (Algorithm 1) sets a bit $S[j]$, where $j \geq z$. and thus $\min\{j \mid h(y, j) = 1\} \geq z$. For any sketch, let δ be the maximum value such that *all* δ least-significant bits of the sketch have been set by the all the items that have been inserted so far. A subsequent insertion is *nil-potent* if it does not reach bit $\delta + 1$, that is, a nil-potent insertion picks a bit which has already been set and thus is guaranteed not to change the sketch. Otherwise, an insertion is *potent*; it sets a bit with

Algorithm 2. SUMINSERT(S,k,c)

```
1:  for i = 1, ..., |S| do
2:    S[i] = 0;
3:  end for
4:  i = 1;
5:  while i ≤ c do
6:    δ = min {z | S[z] = 0};
7:    i = i + (G(1 − 1/2^δ) seeded by ⟨k, c, i⟩);
8:    if i ≤ c then
9:      z = δ + (G(1/2) seeded by ⟨k, c, i⟩);
10:     S[z] = 1;
11:   end if
12: end while
```

index at least $\delta + 1$. Note that a potent insertion does not necessarily alter the sketch, since any bit beyond δ may have also been previously set (but does not belong to the continuous prefix of set bits). To make these points clear, assume that after a number of insertions, the current state of an FM sketch with size $l = 12$ is: 111100101000. The value of δ in this sketch is 4. Consider now a new item y_1 inserted into this sketch using Algorithm 1. If the bit set by this item is one of the first four bits, then y_1 is a nil-potent item. Otherwise, it is called a potent item. Note that if y_1 is potent, it will reach bit five and it will set bit 5 with probability 0.5 (based on Algorithm 1), in which case δ is increased to 5. Alternatively, y_1 may set bits after the fifth one. For example, if it attempts sets the seventh bit to 1, the sketch will not change. On the other hand, it may set the eighthbit to 1, and in this case the new sketch will be: 111100111000. In either of these latter two cases, δ will remain the same.

Now, to develop our algorithm, consider that for FM sketches, an insertion is potent with probability $p = 1/2^{\delta}$, and nil-potent with probability $q = 1 - 1/2^{\delta}$. Thus we can directly decide weather a particular insertion is potent by sampling the Bernoulli distribution with parameter p. If an insertion is deemed nil-potent, it is ignored. Otherwise, we must determine which bit it sets. By the memorylessness of the geometric distribution used in the FM sketches, this is straightforward to emulate: The probability that a potent insertion chooses bit $\delta + i$ is exactly $1/2^i$. Therefore, we can select the value of i by simply drawing from the geometric distribution $\mathbf{G}(1/2)$. We then set bit $\delta + i$, and update δ if appropriate.

To avoid processing each insertion sequentially, we can also compute the run-length of nil-potent insertions directly. Based on the previous discussion, a sequence of nil-potent items corresponds to a sequence of Bernoulli trials with parameter q. Since a sequence of Bernoulli trials defines a geometric distribution with the same parameter, we have that the run-length of nil-potent items in a sequence follows the geometric distribution with parameter q: $\mathbf{G}(1 - 1/2^{\delta})$.

Our algorithm is presented in Algorithm 2, and we recap it now. We initialize the sketch and δ to zero. Then we repeatedly draw from $\mathbf{G}(1-1/2^{\delta})$ to determine

a run-length of nil-potent insertions to skip. We assume that $\mathbf{G}(0)$ is zero (the case of the first item). After each run, we emulate a potent insertion by drawing from $\mathbf{G}(1/2)$ to identify which bit to set. We use the triplet $\langle k, c, i \rangle$ to seed the pseudo-random generator that provides the random values. The reason is that we need to have independent random values every time that we sample from a probability distribution. Since the triplet is unique each time, we are guaranteed that the pseudo-random values that we get are independent of each other.

3.2 Time and Space Bounds

The running time of the algorithm is dominated by the time needed to sample the various geometric distributions. The first consideration is to provide methods to sample the distributions in O(1) expected time and in small space. Then we analyze the overall expected running time of an insertion, and demonstrate that it runs in $O(\log c_i)$ time in expectation and with high probability.

3.2.1 *Sampling the Geometric Distribution.* A key subroutine we use in our algorithm is random sampling from the geometric distribution. If sufficiently precise floating point hardware is available, standard techniques such as the inversion method [Devroye 1986] enable constant-time sampling. Unfortunately, existing sensor motes have neither floating point hardware, nor the capacity to store the sizable lookup tables used by most of the other classic methods (e.g., for binary search). For these reasons, standard methods for drawing from well-known probability distributions are unsuitable. In our earlier work [Considine et al. 2004], our methods necessitated sampling from both the binomial distribution $\mathbf{B}(n, p)$ and the geometric distribution $\mathbf{G}(1 - p)$, but our current methods require only the latter. Implementation of the following algorithmic result enables us to work with table sizes of only 4KB, as we describe later.

LEMMA 2. *Using a precomputed lookup table of size $O(1/p)$, we can sample from $\mathbf{G}(1 - p)$ in expected O(1) time.*

PROOF. Our result leverages the elegant alias method of Walker [1977] that builds lookup tables to efficiently generate random samples from arbitrary probability density functions (pdfs). This work demonstrates the following theorem (which has a simple and beautiful implementation).

THEOREM 3. Walker [1977]. *For any discrete probability density function $\mathcal{D}$ over a sample space of size k, a table of size O(k) can be constructed in O(k) time that enables random variables to be drawn from $\mathcal{D}$ using two table lookups.*

Walker's method applies to finite distributions, but by virtue of the memorylessness property of the geometric distribution, we can apply it to this infinite distribution as well. First we construct a table T of size k in which the first $k - 1$ elements correspond to the probabilities p_i of drawing $1 \leq i \leq k - 1$ from the geometric distribution $\mathbf{G}(1 - p)$ (the body of the distribution), and element k corresponds to the cumulative tail probability of drawing any value larger than or equal to k from $\mathbf{G}(1 - p)$. To draw a random sample from $\mathbf{G}(1 - p)$ we produce a uniform random number $1 \leq i \leq k$. If $i < k$ we simply return $T[i]$

as the next sample from $\mathbf{G}(1-p)$. Otherwise, we need to sample from the tail of the distribution and we iteratively draw another uniform random number, setting the outcome to $T[i]+T[i']$, where i' is the result of the second draw. We continue iteratively until a random draw chooses a value from the body of the distribution. By setting $k=\lceil 1/p\rceil$, we achieve the desired space and time bounds. The space bound is immediate from Walker's theorem. For the time bound, since $1/p$ is the expectation of $\mathbf{G}(1-p)$, the iteration terminates with constant probability after each round, and thus the expected number of rounds is O(1). □

This is easily extended as follows.

LEMMA 3. *Given a fixed set of precomputed tables of size $O(m)$, the geometric distribution $\mathbf{G}(1-1/2^{\delta})$ can be sampled in $O(1+2^{\delta}/m)$ expected time for any non-negative integer δ.*

PROOF. Let $\delta_m=\lceil\log_2 m\rceil$ and construct the tables of Lemma 2 for $p=1/2, 1/4, \ldots, 1/2^{\delta_m}$. These tables have total size $O(2^{\delta_m})=O(m)$ and allow $\mathbf{G}(1-1/2^{\delta})$ to be sampled in $O(1)$ expected time for $\delta\le\delta_m$. For $\delta>\delta_m$, $\mathbf{G}(1-1/2^{\delta})$ can be sampled by repeatedly sampling $\mathbf{G}(1-1/2^{\delta_m})$ until a sample is accepted with probability $1/2^{\delta-\delta_m}$ and the sum of all the samples so far is returned. Since the expected number of samples from $\mathbf{G}(1-1/2^{\delta_m})$ is $2^{\delta-\delta_m}$, the total expected running time is $O(1+2^{\delta-\delta_m})=O(1+2^{\delta}/m)$. □

3.2.2 *Overall Running Time.* We now prove that Algorithm 2 runs in expected time $O(\log c_i)$, a considerable improvement over our previous algorithm [Considine et al. 2004] that runs in $O(\log^2 c_i)$ expected time. Our algorithm selects each potent insertion in turn by sampling from the geometric distribution, then emulates each insertion via another sample from the geometric distribution. Since sampling from the geometric distribution completes in O(1) expected time, it remains to bound the expected number of potent insertions.

LEMMA 4. *Insertion of an element $\langle k_i, c_i\rangle$ generates at most $7\log_2 c_i+6$ potent insertions with probability $1-(3/2)/c$.*

PROOF. In our method, each potent insertion increases the value of δ by at least one with probability 1/2. The first $6\log_2 c_i$ potent insertions increase δ an expected $3\log_2 c_i$ times. Using Chernoff bounds, the probability that the first $6\log_2 c_i$ potent insertions increase δ fewer than $\log_2 c_i$ times is

$$\left(\frac{e^{-2/3}}{(1-2/3)^{(1-2/3)}}\right)^{3\log_2 c_i}\le 1/2^{\log_2 c_i}\le 1/c_i.$$

After the first $6\log_2 c_i$ potent insertions, $\delta\ge\log_2 c_i$ with probability $\ge 1-1/c$.

If the method has not yet terminated once $\delta\ge\log_2 c_i$, any potent insertion must have reached bit $\log_2 c_i$, which occurs with probability $1/c_i$. The expected number of insertions reaching bit $\log_2 c_i$ within the next c_i insertions is one. Using Chernoff bounds, the probability that there are more than $\log_2 c+6$

insertions reaching bit $\log_2 c_i$ within the next c_i insertions is bounded by

$$\left(\frac{e^{\log_2 c_i+5}}{(1+\log_2 c_i+5)^{(1+\log_2 c_i)}}\right)^1 \leq 1/2^{1+\log_2 ci} \leq 1/2c_i.$$

This also bounds the number of potent insertions, so there are at most $\log_2 c+6$ more potent insertions with probability greater than or equal to $1/2c_i$. □

It remains to bound the time within each iteration of the outer loop. Maintaining δ is trivially done in $O(1)$ amortized time per iteration. The time to process an insertion of a potent item in steps 9 and 10 takes also $O(1)$ time per iteration, since sampling from $G(1/2)$ takes constant time. Applying Lemma 3 allows us to analyze the total cost of sampling and bound the running time of Algorithm 2.

THEOREM 4. *Given a set of lookup tables of size $O(m)$, any element $\langle k_i, c_i\rangle$ can be inserted into a sum sketch in $O(\log c_i + c_i/m)$ expected time.*

PROOF. Let $\delta_m = \log_2 m$ and construct the lookup tables as in Lemma 3. As long as $\delta \leq \delta_m$, then sampling $\mathbf{G}(1-1/2^\delta)$ takes $O(1)$ expected time. At this point, there are two main cases to consider, depending on whether Algorithm 2 completes before δ exceeds δ_m.

Before continuing, note that δ is bounded by the number of iterations of the outer loop (since δ bits must be set to one first), which in turn is $O(\log c_i)$ with high probability by Lemma 4. In the first case where $\delta \leq \delta_m$ at completion, then each iteration of the outer loop takes expected time $O(1)$, and the total running time is $O(\delta) = O(\log c_i)$.

We now focus on the second and more involved case where $\delta > \delta_m$ at completion. Since δ increases by at least one with probability $1/2$ for each iteration of the outer loop, the expected number of rounds and the expected time before δ exceeds δ_m is $O(\delta_m)$. From this point, the construction in the proof of Lemma 3 is used and sampling $\mathbf{G}(1-1/2^\delta)$ will involve repeated samples of $\mathbf{G}(1-2^{\delta_m})$. Each of these samples takes $O(1)$ expected time and corresponds to increasing the variable j by 2^{δ_m} in expectation. Thus, as long as j is $O(c_i)$ at the end of Algorithm 2, then the expected number of samples to $\mathbf{G}(1-2^{\delta_m})$ and the expected time after δ exceeds δ_m is $O(c_i/2^{\delta_m}) = O(c_i/m)$. But what if j is not $O(c_i)$? We can avoid this case by noting that the precise value of j is not important once it is known that j will exceed c_i and modifying the sampling procedure to stop early once this is the case. Thus, the total expected time is $O(\log c_i)$ for the outer loop and earlier samples, plus $O(c_i/m)$ for extra sampling costs if δ exceeds δ_m, for a total of $O(\log c_i + c_i/m)$. □

For example, if c is an upper bound on the c_i values, then insertions are possible in $O(\log c_i)$ expected time using $O(c/\log c)$ space, or $O(\log^2 c_i)$ expected time using $O(c/\log^2 c)$ space. The latter bound was achieved by the earlier version of this article [Considine et al. 2004]. We note that for small c_i values, it may be faster to use a hybrid implementation combining the naive and scalable insertion functions. Especially for very low c_i values, the naive insertion function will be faster.

3.3 Improving Accuracy and Other Estimators and Aggregates

To improve the accuracy and confidence of the estimator we use multiple bitmaps similarly to the FM sketch for distinct counting. PCSA can also be applied to summation sketches, but greater care is needed. The potential for imbalance is much larger with summation sketches; a single item can contribute an arbitrarily large fraction of the total sum. Thus, we employ the following strategy. Each c_i value has the form $c_i = q_i b + r_i$ for some integers q_i and r_i, with $0 \leq r_i < b$. We add r_i distinct items only to one bitmap as in standard PCSA, and then add q_i to all bitmaps independently. Thus, we preserve the balance necessary for the improved accuracy and its analysis, but at the cost of $O(b \log(c_i/b))$ for each insertion. We employ PCSA in our experiments.

The range-efficient sketch algorithm [Aduri and Tirthapura 2005] can be used to solve the distinct summation problem that we discussed here with logarithmic update time. However, this method requires computations in an algebraic field using modular multiplications and divisions with extremely large values. To perform these operations in a small sensor device with limited hardware is a challenge. In practice, a range-efficient sketch implemented on a sensor device will be slower than the simple FM sketch that uses the linear-time insertion method, because of the cost of these operations and the need to avoid overflows.

In situations where computational resources are severely constrained, it may be desirable to reduce the cost of performing insertion operations with summation sketches. We now briefly mention some trade-offs in the computational time at the cost of increased communication and decreased accuracy. While this is unlikely to be desirable in sensor networks, given the high power costs of communication relative to computation, it may be desirable in other settings where there are large numbers of items per node.

Suppose that the largest value being inserted is bounded by y^x. Then, instead of using the previous approach, We can do the following: We can use x different summation sketches, each corresponding to a different digit of the c_i's using radix y. To add a c_i value, each digit of c_i is inserted into the corresponding sketch, taking expected $O(x \log y)$ time, and estimates are made by summing the counting sketch estimates with the appropriate weights. The accuracy of this approach is essentially the same as before, with an increase in space bounded by a factor of x.

Of particular note is the case $x = 2$, which preserves the asymptotic sketch size and insertion costs, but dramatically drops the necessary table sizes (to approximately the square root of the old sizes). However, doubling the sketch size and associated transmission costs seems undesirable, except when local storage and processing capability are extremely limited in comparison. Similarly, the insertion time can be further dropped to $O(z \log y)$ expected time without affecting space requirements, if only the z most significant nonzero digits are inserted, but the expected estimate may be too low by a factor of $(1 + (y - 1)/(y^z - 1))$.

So far, we have only discussed two aggregates, COUNT and SUM, and their approximation with non-negative integer inputs. These techniques can also be

extended to other aggregate functions beyond summation and counting. For example, AVG can also be computed directly from COUNT and SUM sketches. The second moment can be computed as an average of the squares of the items, and then combined with the average of the items, to compute the variance and standard deviation.

While this section has focused upon approximating the sum of integers, these techniques can be extended to other data types. For example, the extension to handle fixed point numbers is trivial. Similarly, products can be approximated by summing logarithms in fixed point notation, with a corresponding loosening of the error bounds. Summing floating point numbers is also possible at greater expense, but we do not provide full details here.

A natural question is how to handle duplicate keys with different values. Algorithm 2 will account for only distinct key/value pairs. The sketch can also be modified to account only the maximum value associated per key, by seeding the hash functions using only k_i and not c_i. This value is known as the *max-dominance norm* [Cormode et al. 2003; Stoev et al. 2007].

4. FREQUENCY COUNTING SKETCHES

We now discuss how the FM sketches can be used to make the Count-Min (CM) sketch duplicate-insensitive and hence suitable for multipath in-network aggregation. The CM sketch accurately estimates the frequency of appearances of stream elements in one pass and with small space proposed by Cormode and Muthukrishnan [2005a]. In a subsequent work [Cormode and Muthukrishnan 2005b], the same authors extended the CM with FM sketches as an example of a cascading sketch (composition of a sketch inside another sketch). In this section, we provide the description of the CMFM sketch and we provide a new and more detailed analysis of the error and failure probabilities of the CMFM sketch.

4.1 CM Sketch Basics

Let $M = \{x_1, x_2, x_3, \dots\}$ denote a multiset of elements from a domain $[1, D]$. The CM sketch is a simple randomized data structure that can estimate the frequency of any element x in M, and consists of a $k \times d$ matrix of counters.

$$CM = \begin{bmatrix} c_{1,1} & c_{1,2} & \dots & c_{1,d} \\ c_{2,1} & c_{2,2} & \dots & c_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ c_{k,1} & c_{k,2} & \dots & c_{k,d} \end{bmatrix}$$

To estimate the frequency of a given element, the CM sketch uses a simple hashing scheme with a set of k independent hash functions $h_1, \dots, h_k : [D] \to [d]$, one hash function per row of the matrix. Each hash function maps a given element to a specific counter of the filter on a specific row. The matrix is initialized by setting all counters to zero. Then, $h_i(\cdot)$ is evaluated for all $x \in M$, and the counters $CM_x[i], 1 \le i \le k$ (where $CM_x[i] = \{CM[i, j] : j = h_i(x)\}$) are increased by 1. If the hash functions are independent, then the probability that two elements will hash to exactly the same set of counters is equal to $\frac{1}{d^k}$.

Given this structure, the estimated frequency of x is given by the formula $|\widehat{x}| = \min_i \{CM_x[i]\}$. Intuitively, the minimum of any counter yields a frequency estimate that is the least affected by hash function collisions of other elements on the same counters. The CM sketch imposes specific constraints on k and d, in order to give tight guarantees on the estimation accuracy. In particular, the following theorem holds.

THEOREM 5. *For user-defined constants $\epsilon, \delta \in (0, 1]$, and a CM sketch with $d = \lceil e/\epsilon \rceil$, $k = \lceil \ln(1/\delta) \rceil$ let $|x|$ be the exact frequency of element $x \in M$, and $|\widehat{x}|$ be the estimated frequency computed using the sketch. Then, $|\widehat{x}| \geq |x|$, and with probability $1 - \delta : |\widehat{x}| \leq |x| + \epsilon |M|$.*

Note that in the preceding theorem, $|M| = \sum_{i=1}^{D} |x_i|$, where $|x_i|$ is the frequency of element x_i in M. It can be shown that the CM sketch has the following properties.

PROPERTY 3. *The CM sketch of the union of two multisets is the simple matrix addition of their individual CM sketches; that is,* $\mathrm{CM}(M_1 \cup M_2)[i, j] = \mathrm{CM}(M_1)[i, j] + \mathrm{CM}(M_2)[i, j]$.

PROPERTY 4. *The CM sketch is order-insensitive.* $\mathrm{CM}(M)$ *is entirely determined by the items of M. Ordering of insertions does not affect the structure.*

4.2 Design and Theory of Duplicate-Insensitive CM Sketches

Apart from order insensitivity, we would ideally like the CM sketch to be duplicate-insensitive in terms of the union operation. Notice that double-counting a certain CM sketch into a union operation many times will yield errors in estimated frequencies that grow with the total number of duplicate additions that occurred. A similar effect occurs when double-counting duplicate elements that have been inserted across different CM sketches.

The simplest approach to make CM sketches duplicate-insensitive is to replace every matrix counter with an FM sketch, and take advantage of the duplicate insensitivity of these sketches [Cormode and Muthukrishnan 2005b]. Essentially, we use the FM sketches to estimate the magnitude of each counter, that is, to estimate how many elements have been added to the specific FM sketch. We call this construction the CMFM sketch, for which the following properties hold.

PROPERTY 5. *The CMFM sketch of the union of two multisets is the simple matrix addition of their individual CMFM sketches, where each element addition is given by the bit-wise OR of the corresponding FM sketches; that is,* $\mathrm{CMFM}(M_1 \cup M_2)[i, j] = \mathrm{CMFM}(M_1)[i, j] \vee \mathrm{CMFM}(M_2)[i, j]$.

PROPERTY 6. *The CMFM sketch is duplicate-insensitive.*

Given the structure of the CMFM, we need to theoretically derive the error bounds for the estimated frequencies, based on the individual errors introduced both by the FM sketch approximations and the CM sketch hash function collisions. To do this, we need to use the following theorem about the error

guarantees of the FM sketches that can be derived from Bar-Yossef et al. [2002] and Ganguly et al. [2003].

THEOREM 6. *Given parameters ϵ and δ, we can create an FM sketch with size $O(\frac{1}{\epsilon^2}\log(1/\delta)\log(N))$ bits, that can report the number of N unique values in a multiset M with error at most ϵN and probability of failure at most δ. The FM sketch can be updated in time $O(\log(1/\delta))$.*

Based on the preceding theorem, next we show the following bounds on the error of the CMFM sketch.

THEOREM 7. *For a CMFM sketch with $d = \lceil(1+\epsilon_f)e/\epsilon_c\rceil$, $k = \lceil\ln(1/\delta_c)\rceil$ and user-defined constants $\epsilon_f, \epsilon_c, \delta_f, \delta_c$, let $|x|$ be the exact frequency of element $x \in M$, and $|\widehat{x}|$ be the estimated frequency computed using the sketch. Then, with probability at least $(1-\delta_f)^k : |\widehat{x}| \geq (1-\epsilon_f)|x|$ and with probability at least $(1-\delta_f)^k(1-\delta_c) : |\widehat{x}| \leq (1+\epsilon_f)|x| + \epsilon_c|M|$.*

PROOF. The lower bound can be proven as follows. Let $X[i]$ denote the set of elements that hash to exactly the same FM sketch as element x on row i. Then, $|X[i]|$ quantifies the total number of insertions to FM sketch $CM_x[i]$, which implies that by construction the FM sketch $CM_x[i]$ gives the estimate $|\widehat{X[i]}|$. Also, let $|x|$ denote the total number of insertions of x in the sketch. Clearly, $|X[i]|$ accounts for the $|x|$ insertions plus the collisions of other elements to the same FM. Let the estimation accuracy of each individual FM sketch be bounded by

$$(1-\epsilon_f)|X[i]| \leq |\widehat{X[i]}| \leq (1+\epsilon_f)|X[i]|$$

with probability $1-\delta_f$ (Theorem 6). Then,

$$\begin{aligned} CM_x[i] &\geq (1-\epsilon_f)|X[i]| \\ &\geq (1-\epsilon_f)|x| \end{aligned}$$

with probability $1-\delta_f$. By using

$$|\widehat{x}| = \min_{i=1}^{k}\{CM_x[i]\}$$

as the estimation function, the lower bound follows. Notice that the lower bound succeeds with probability $(1-\delta_f)^k$, namely, the probability that the estimates of all FM sketches will be correct.

To prove the upper bound, assuming perfect hash functions it holds that

$$P = Pr[h_i(x) = h_i(y)] \leq \frac{1}{d}.$$

Let $|Y[i]|$ denote the total number of collisions of elements other than x on $CM_x[i]$. Then

$$E\{|Y[i]|\} = \sum_{y\neq x} P \cdot y \Rightarrow E\{|Y[i]|\} \leq \frac{1}{d}|M|.$$

Thus, by using $Pr[A > B] \leq Pr[A' > B]$, for $A < A'$, and Markov's inequality,

$$Pr[|\widehat{x}| > (1+\epsilon_f)|x| + \epsilon_c|M|] =$$

$$Pr[\forall i, CM_x[i] > (1+\epsilon_f)|x| + \epsilon_c|M|]$$

$$= Pr[\forall i, |\widehat{X[i]}| > (1+\epsilon_f)|x| + \epsilon_c|M|]$$

$$\leq Pr[\forall i, (1+\epsilon_f)|X[i]| > (1+\epsilon_f)|x| + \epsilon_c|M|]$$

$$\leq Pr[\forall i, (1+\epsilon_f)(|x| + |Y[i]|) > (1+\epsilon_f)|x| + \epsilon_c d\, E\{|Y[i]|\}]$$

$$= Pr[\forall i, |Y[i]| > \frac{\epsilon_c d}{1+\epsilon_f} E\{|Y[i]|\}] \leq \left(\frac{1+\epsilon_f}{\epsilon_c d}\right)^k.$$

Now, by setting $e = (\epsilon_c d)/(1+\epsilon_f)$, we get

$$Pr[|\widehat{x}| \leq (1+\epsilon_f)|x| + \epsilon_c|M|] > 1 - e^{-k}.$$

Thus, $\delta_c = e^{-k} \Rightarrow k = \ln(1/\delta_c)$. The preceding holds only if none of the FM sketches fails, thus the estimation will succeed with probability $(1-\delta_f)^k$ $(1-\delta_c)$. □

Since the success of the CMFM sketch depends heavily on the success of the individual FM sketches, for the given analysis, it is essential to guarantee that the failure probability of FM sketches is low, in order to limit the factor of the exponent k. In practice, for a failure probability $\delta_c = 1\%$ the CM sketch requires $k = \lceil \ln(1/0.01) \rceil = 5$ hash functions. If the FM sketches fail with probability $\delta_f = 1\%$ as well, then the total probability of failure for the CMFM sketch, from Theorem 7, is 5% and 6% for the lower and upper bound, respectively.

5. QUANTILE ESTIMATION SKETCHES

We now demonstrate how to extend a quantile estimation technique to be duplicate-insensitive for use in sensor network environments. The Quantile Digest (QD) [Shrivastava et al. 2004] is an efficient ϵ-approximation technique for estimating arbitrary quantiles over data produced by a number of sensors. The idea is to maintain a small summary at each sensor, and to combine individual summaries in order to answer quantile queries on the union of the data, with relative error at most ϵ.

5.1 Quantile Digest Basics

Given a multiset M with values in $[1, D]$ and a fraction $q \in (0, 1)$, the qth quantile of M is the value with rank $\lceil q|M| \rceil$ in the sorted sequence of values in M (the MEDIAN is a special quantile, with $q = 0.5$). If M is known in full, then quantile computation can be done very efficiently; for example, by the classical linear-time algorithm [Floyd and Rivest 1975]. On the other hand, if M is distributed over a number of sources, or if the size of M is too large to be retained in full (e.g., in streaming applications), then exact computation of quantiles is not an easy problem. In the first case, special distributed algorithms need to be used [Greenwald and Khanna 2004]. In the second case, exact quantile computation is not feasible at all [Munro and Paterson 1980]. For these reasons, various approximation algorithms that work by summarizing the data

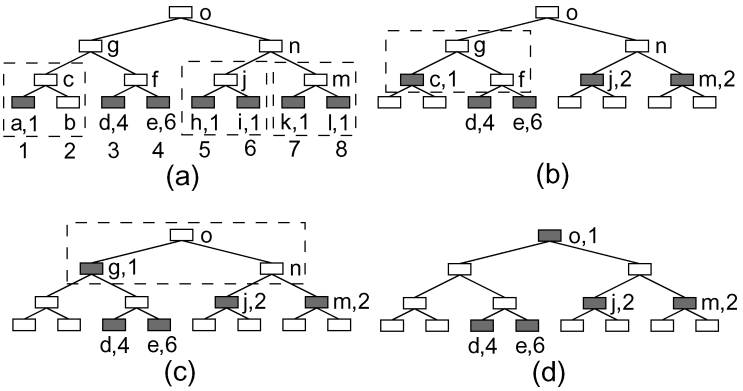


Fig. 2. Building a quantile digest. The complete conceptual tree is shown here. Gray nodes correspond to the actual sketch. For this example $k = 5$.

appropriately have been proposed for quantile computation. We say that an algorithm computes an ϵ-approximate quantile if the estimated answer has rank between $\lceil (q - \epsilon)|M| \rceil$ and $\lceil (q + \epsilon)|M| \rceil$.

QD uses a predefined *binary tree structure* to capture the distribution of values in M over the domain $[1, D]$. Every node of the tree is associated with a tuple $\langle N, R_N, C_N \rangle$, where N is a unique node identifier, R_N is a range of values from $[1, D]$ (where R_N^{min}, R_N^{max} will be used to denote the upper and lower bound of the range, respectively), and C_N is a tally of the number of elements of M with values in R_N. Values are assigned to leaves in a sorted order from left to right. Nodes in higher levels of the tree correspond to *dyadic ranges* over the values of the lower levels. The root of the tree corresponds to the whole domain. In addition, node identifiers are assigned by following a sequence that is equivalent with a postorder traversal of the tree. An example is shown in Figure 2(a), for the domain of values [1, 8]. The identifiers of the nodes and the frequency of each value are also shown.

This binary tree is *conceptually* maintained locally by all the sensors. In practice, every sensor needs to instantiate only a subset of the tree nodes at a time, and transmit this subset to the destination. The subset constitutes the actual QD sketch. Notice that a QD sketch that consists of all the leaves is equivalent to an equiwidth histogram with bucket size 1, which can be used to provide exact solutions. To guarantee approximate answers within an error threshold and decrease the size of the sketch we need to choose which nodes to instantiate and include in the QD. This is governed by the following two conditions:

$$C_N \leq \lfloor |M|/k \rfloor, \text{if } N \text{ is not a leaf} \tag{1}$$

$$C_N + C_{N_p} + C_{N_s} > \lfloor |M|/k \rfloor \tag{2}$$

where N, N_p, and N_s are a node, the node's parent, and its sibling, respectively, and k is a user-defined parameter that controls the accuracy and the compression ratio of the sketch. Eq. (1) can be used to guarantee special error bounds on QD, since every node cannot increase the error of the estimation by more

than $\lfloor |M|/k \rfloor$. Eq. (2) implies that if two adjacent leaves have a low count, they are compressed to save space by preserving only their parent.

A QD is constructed as follows. First, we compute the frequency C_N of every leaf. We do not instantiate any index nodes, but we assume that for these nodes $C_N = 0$. Next, we propagate values up the tree as follows: If a leaf and its sibling violate Eq. (2) we merge the two leaves, instantiate their parent, set $C_{N_p} = C_N + C_{N_s}$, and discard N, N_s. We continue the same process left to right and bottom to top, until no more nodes can be merged. At the end, all nodes that have not been discarded constitute the QD. An example is shown in Figures 2(a)–(d) for $|M| = 15$ and $k = 5$. The quantile digest at each step of the compression algorithm consists of the gray nodes. The final QD sketch is the following: $\langle d, 4\rangle$, $\langle e, 6\rangle$, $\langle j, 2\rangle$, $\langle m, 2\rangle$, $\langle o, 1\rangle$.

Merging QDs is straightforward. Starting from the complete conceptual binary tree with all counters initialized to 0, incoming QD nodes from other sources are added to their respective counters in the tree. Then, Eqs. (1) and (2) are applied on the resulting tree, where $|M| = |M_1| + |M_2| + \ldots$ is the cardinality of the union of the sets corresponding to the incoming QDs. Answering quantile queries on any QD requires first sorting the nodes in increasing identifier order. Now, given $q \in (0, 1)$, we start adding node counters from left to right in the sorted order, until the sum becomes larger than $\lceil q|M| \rceil$, after some node N, at which point we know that at least $\lceil q|M| \rceil$ values are included in the range $[1, R_N^{max}]$. For the qth quantile of M we report the value R_N^{max}. In the previous example the computation yields node e with value 4 as the estimated MEDIAN.

The following hold for quantile digests [Shrivastava et al. 2004].

LEMMA 5. *For a QD with compression factor k, the maximum error in C_N for any node is $\frac{\log_2(D)}{k}|M|$.*

THEOREM 8. *A QD Q can answer any quantile query with error ϵ_q such that $\epsilon_q \leq \frac{3\log_2(D)}{|Q|}$, where $|Q|$, the number of nodes in the digest, satisfies $|Q| < 3k$, given compression factor k.*

PROPERTY 7. *Given n QDs: $QD_1, \ldots, QD_n$, built on multisets of values $M_1, \ldots, M_n$, each with maximum relative error ϵ_q, the merging algorithm combines them into a QD for $M_\cup = M_1 \cup \ldots \cup M_n$ with the same relative error.*

5.2 Duplicate Insensitivity

The QD sketch is clearly not duplicate-insensitive. For example, merging $QD(M)$ with itself produces a new QD for the larger multiset $M \cup M$. As before, to distinguish between true duplicate values introduced by multipath routing and values reported by different sensors that happen to be identical we create a triplet for each value in each sensor (*sensor id, value id, value*). (Here, sensors may report multiple values, thus the use of multisets, and where the multiset in each sensor is a set of triplets.) The goal now is to compute the quantiles using the values in all triplets, but with each triplet contributing exactly once.

Algorithm 3. CompressQD(Q, m, k)

```
1:  l = log(D) − 1 // l initialized to the leaf level;
2:  while l > 0 do
3:    for all u in level l do
4:      Compute FM_t = Merge(FM_u, FM_up, FM_us)
5:      if FM_t < ⌊m/k⌋ then
6:        FM_up = FM_t
7:        delete u and us
8:      end if
9:    end for
10:   l = l − 1
11: end while
```

We can again utilize FM sketches to make a combined QDFM sketch that is duplicate-insensitive. We initially create the leaf nodes of the QD by using an FM sketch for all nonempty nodes and inserting each value once to the corresponding sketch. Then, we use the Algorithm 3 [Shrivastava et al. 2004] to compress the digest. In this algorithm, m is the total number of values in the digest and k the compression ratio. We process the nodes level by level. For each node in a given level, we first merge its FM sketch with the FM sketches of its parent (up) and sibling node (us). Then, we use the sketch to estimate the total number of items in the three nodes and if it is smaller than $\lfloor \frac{m}{k} \rfloor$, we merge the u and us with the parent node. This continues until it reaches the root level. To merge two (or more) digests, we merge the FM sketches of the same nodes in the trees and then we apply the CompressQD algorithm. Note that merging a QDFM sketch with itself produces the original QDFM sketch. Also, if duplicate values are inserted in the FM sketches across sensors, they will be counted only once. Therefore we have the next property.

Property 8. *The QDFM sketch is duplicate-insensitive.*

Let a QDFM sketch $\{\langle A, FM_A\rangle, \langle B, FM_B\rangle, \dots\}$, where each FM_N is an estimator for C_N, with $(1-\epsilon_f)C_N \leq \widehat{C}_N \leq (1+\epsilon_f)C_N$. To find the qth quantile of M, we iteratively combine FM sketches from left to right (in the sorted identifier order) using the OR operation on the bit vectors, until $\widehat{C}_{A\cup\dots\cup N} \geq q|M|$ (where $\widehat{C}_{A\cup\dots\cup N}$ is the estimation produced by the combined FM sketch $FM_A \cup \dots \cup FM_N$). We report as the qth quantile of M, the value R_N^{max}.

Unfortunately, we cannot give the same analytical guarantees as the original QD sketch. The main reason is that because of the existence of duplicate triplets when we merge two QDFMs, we may create a new node that does not satisfy Eq. (1). Therefore, we can use the QDFM as a heuristic to estimate quantiles inside a sensor network. Since it does not provide error guarantees as the original QD, it should be used with care and only when other alternatives are not possible or are too expensive for use in practice.

6. EXPERIMENTAL EVALUATION

In this section, we present an evaluation of our methods using the TAG simulator [Madden et al. 2002]. In the TAG simulator, each node aggregates results received from its children with its own reading, and then sends the aggregate to one or more of its parents. Any node within broadcast range which is at a lower level (closer to the root) is considered a candidate parent. For our basic experimental network topology we used a regular 30×30 grid with 900 sensors. The communication radius allowed the nearest eight grid neighbors to be within transmission range, and the default link loss rate was 5%. The root node was always at the center of the grid. Figure 1 illustrates an example of 7×7 grid. In addition, we extended the basic network topology to more realistic settings, using road networks taken from the U.S. Census Bureau TIGER/Line datasets [U.S. Census Bureau 2009]. We randomly placed sensors on the road network by using various distributions described next. First, we evaluate the performance of the FM sketches for *count* and *sum* aggregate queries. Then, we evaluate the CMFM and QDFM sketches for frequency counting and quantile estimation queries, respectively.

6.1 Count and Sum Queries

We implemented and compared the following methods.

—*TAG1:* the main strategy of TAG [Madden et al. 2002] (each sensor sends its aggregate to a single parent).
—*TAG2:* the "fractional parents" strategy of TAG [Madden et al. 2002] described in Section 2.2.
—*SKETCH:* the strategy described in Section 2.3 using our proposed duplicate-insensitive sketches.
—*LIST:* the aggregate consists of an explicit list of all the items in the aggregate with any duplicates removed. These lists are sent to all parents.

TAG1 and TAG2 are previous competitors. LIST is a straightforward solution. For all results we average over a total of 500 simulations and present in the graphs the mean along with the 5th and 95th percentiles.

We performed simulations for estimating the total number of active sensors (a *count* query) and the sum of values reported by these sensors (a *sum* query where sensors produce values in [1, 100] uniformly at random), on the regular grid topology. We evaluated all of the aforementioned aggregation strategies in order to expose their advantages and disadvantages.

First, we measured the approximation accuracy of the proposed sketches as a function of the maximum message size per transmission. In addition, we measured the communication overhead of these approaches in order to assess the accuracy versus power-consumption trade-offs. Then, we evaluated the robustness of all aggregation strategies under varying link loss and node loss rates. We also performed scaling experiments for increasing sensor network sizes. Finally, we ran the same experiments on more realistic synthetic topologies, rather than regular grids.

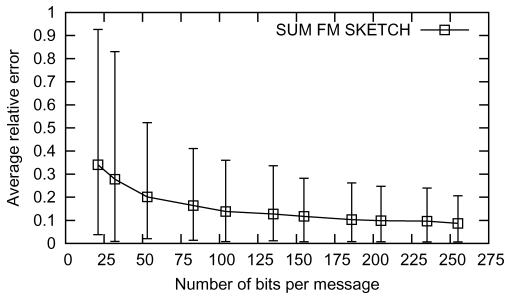


Fig. 3. Estimation accuracy of summation FM sketch as a function of sketch size.

Table I. Communication Cost for *Sum* Queries

Strategy	Total Bytes	Messages Sent	Messages Received
TAG1	1800	900	900
TAG2	1800	900	2468
SUM FM SKETCH	10843	900	2468
LIST	170424	900	2468

The results for the approximation accuracy of FM sketches as a function of sketch size for a *sum* query are shown in Figure 3. We run a one-shot *sum* query that aggregates the values reported by all the sensors during one epoch and average over 500 runs. We use the compression technique of Palmer et al. [2002] that reduced the size of the estimator by about 60% to 70%. As expected, the larger the sketch size (hence, the larger the number of bitmaps per FM sketch), the smaller the relative error observed for the estimated query results. These results are in line with the guarantees that we get for the analysis of the FM sketches. The standard error that we get from the experiments is a little bit better than the expected $0.78/\sqrt{m}$. In the rest, we use 20 bitmaps of 16 bits each for the FM sketches (approximately 100 bits per compressed FM sketch), which provides a good accuracy versus communication-overhead trade-off for our setting.

Table I shows the total number of bytes transmitted along with the total number of messages sent and received (assuming no losses). For the TAG1 and TAG2 approaches it is assumed that every value transmitted is 16 bits. LIST sends a number of $\langle id, value \rangle$ pairs per message, using 32 bits per pair (two bytes for the id and two bytes for the associated value). As expected, the TAG strategies send the least amount of data, while LIST is the most expensive. The SKETCH strategy has approximately 6 times larger communication overhead than TAG, and 16 times smaller than LIST. Notice here that the communication cost of the SKETCH strategy can be further decreased at the cost of approximation accuracy by reducing the number of bitmaps used, as Figure 3 illustrates. Nevertheless, the SKETCH strategy leverages improved approximation quality for somewhat increased communication cost when compared with TAG, especially for low to medium loss rates, as will become clear shortly. Very similar observations were made for *count* queries, hence these results are omitted.

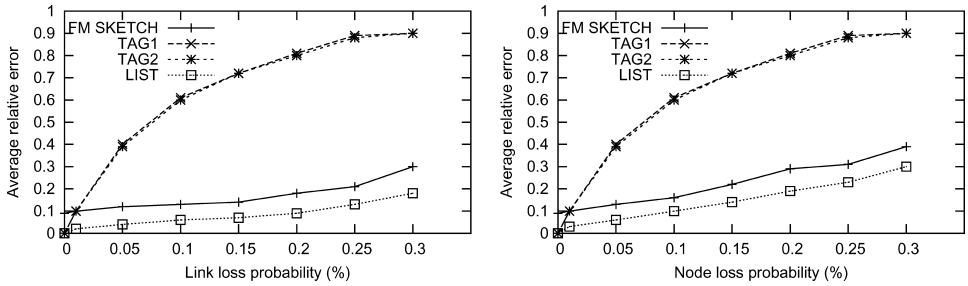


Fig. 4. Performance of aggregation strategies as a function of link and node loss rate.

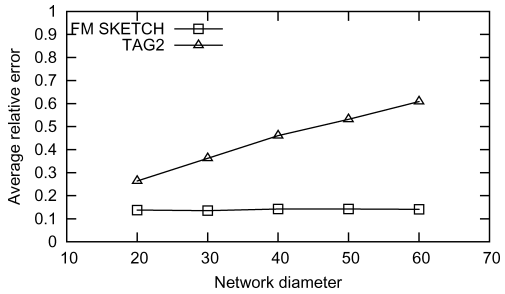


Fig. 5. Performance of aggregation strategies for varying network sizes.

Next, we evaluate the robustness of the four aggregation strategies as a function of link loss rates. We use a uniform probability of failure for all the links, meaning that a message might fail to be delivered to its intended recipient with a preset probability. We evaluate a *count* query in this experiment. Notice here that the exact answer to this should be equal to the number of sensors in the grid, namely, 900.

Figure 4(a) plots the results. We can see that the SKETCH and LIST strategies outperform the TAG strategies for low, medium, and high loss rates. Only when the loss rate is close to zero are TAG approaches better than the SKETCH, since these approaches provide exact results whereas the SKETCH provides approximate results. As the loss rate increases, the answers returned by LIST and SKETCH strategies degrade slowly, while those returned by the TAG approaches deteriorate at a much higher rate. It should be noted here that the answers returned by the SKETCH strategies are different than the results of the LIST approach because of the use of FM sketches.

Figure 4(b) shows the effect of node losses assuming 5% loss rate. The general trend is similar to those for link losses, with the only difference being that the average estimated counts reported decrease at a faster rate. Intuitively, this happens due to the fact that if a node fails its value is certainly lost, while for link failures *all* potential upstream links of a specific node must fail for its value to be lost.

We also tested the aggregation strategies for sensor networks of varying sizes. Figure 5 plots the results. Despite the fact that the loss rate is being held constant, the TAG strategies perform poorly as the network size increases.

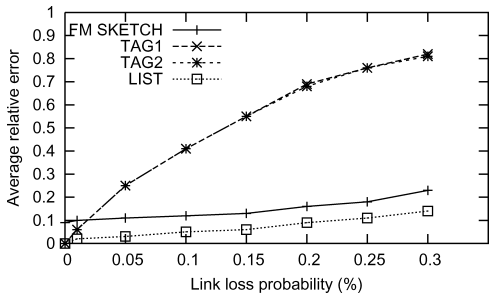
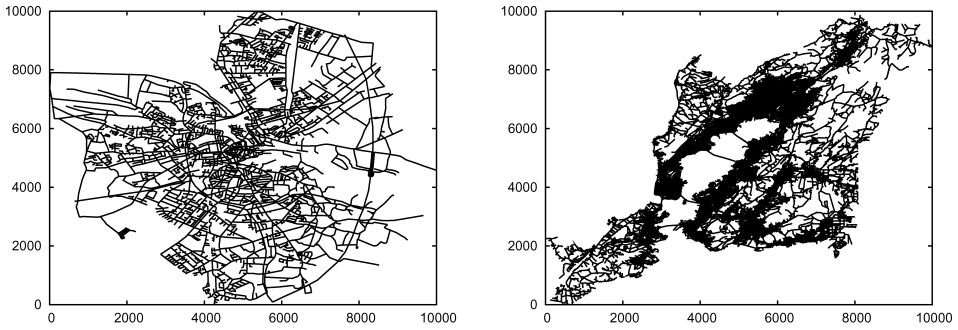


Fig. 6. Performance of aggregation strategies for random sensor placements as a function of link loss rates.



(a) Oldenburg (6105 nodes) (b) San Francisco (174956 nodes)

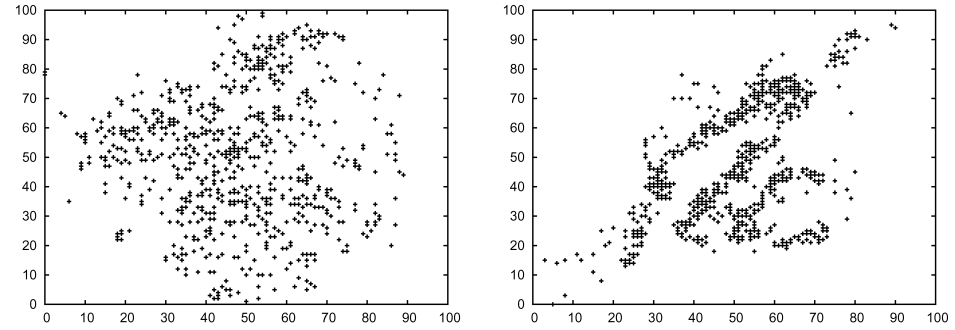
Fig. 7. Road networks used in our experiments.

Meanwhile, the FM sketch maintains an almost constant average relative error, around 13%. Similar results were observed for sum queries.

The next experiment evaluates the different strategies for random sensor placements. For these experiments we doubled the communication radius to compensate for sparse regions of connectivity. Figure 6 shows the estimated *count* as a function of link loss rate. Clearly, the results show that the basic trends of all strategies remain similar to the ones observed for uniform grid placements.

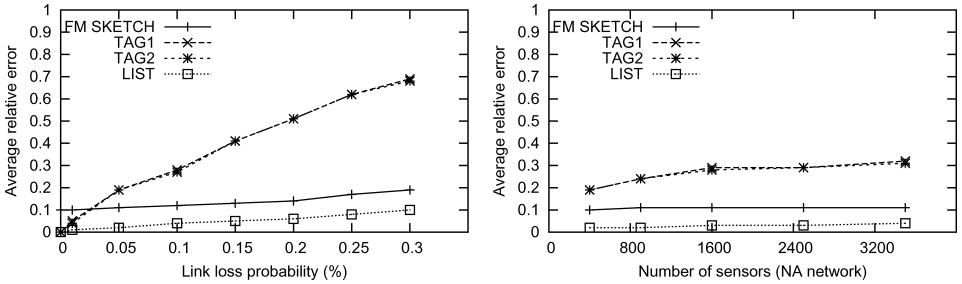
Finally, we replace the simplistic network topology discussed before with more realistic settings, namely with sensors that are placed on real road networks. Given a road network, a number of data points are generated uniformly at random on the graph's edges. The goal is to simulate possible sensor locations on traffic lights and other sites. For all experiments we used a large enough communication radius in order to ensure the connectivity of all sensors in the network. Some real road networks used in our experiments are shown in Figures 7(a) and 7(b), from various metropolitan areas. Placements of 900 sensors on these road networks are shown in Figures 8(a) and 8(b).

We performed the same set of experiments for the real road networks as with the grid topology, observing very similar trends in all cases. Indicatively, in Figure 9(a) we show the results for the performance of all aggregation strategies as a function of link loss rates for *count* queries. Once again, the clear winners



(a) Oldenberg (b) San Francisco

Fig. 8. The placement of 900 sensors on these networks.



(a) Estimated *count* as a function of link loss rate

(b) Estimated *count* as a function of number of sensors

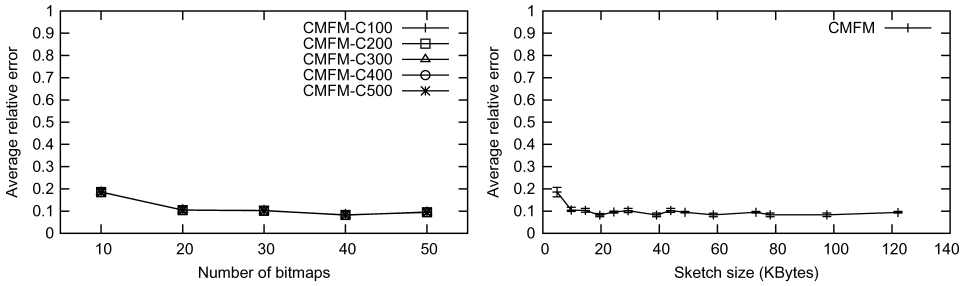
Fig. 9. Experiments on road network topologies.

are the SKETCH-based strategies and LIST, estimating the number of sensors with decent accuracy even for low to medium link losses. Figure 9(b) presents a scalability experiment for increasing the network size from 400 to 3600 nodes while holding link loss rate equal to 5%. The results are similar to the grid topology experiment.

6.2 Frequency Counting

We turn our attention to duplicate-insensitive sketch constructions for frequency estimation. We use the CMFM sketch for estimating value frequencies in a sensor network. Once again we use a 30×30 uniform grid of sensors. We evaluate the accuracy of the sketches by estimating the frequency of the top 25 most frequent data values produced by all the sensors during one epoch and averaging the results over all epochs (for a total of 30 epochs). Each sensor produces 1000 values per epoch in range [0, 100000] using a Zipf distribution with skew equal to 1.2.

The only available alternative for estimating frequencies on a sensor network is the random sampling approach [Nath et al. 2004]. Random sampling is not expected to work well for estimating element frequencies. In our experiments we implemented this approach and ran it for all simulations. The average



(a) average error as a function of the number of bitmaps per FM

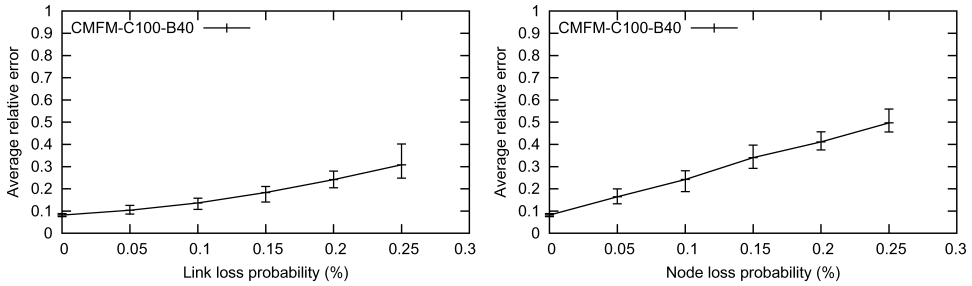
(b) average error as a function of sketch size

Fig. 10. Frequency estimation error as a function of sketch size.

error of the technique was consistently above 90%, hence we refrain here from including it in the graphs for clarity. Other approaches in Nath et al. [2004] have been proposed for identifying heavy hitters, but these are not as general as frequency counting and we refrain from delving into further details.

First, we evaluate the accuracy of the CMFM sketch as a function of sketch size. The size of the sketch is equal to the number of hashes times the number of FM sketches per hash times the size of each FM sketch. For this experiment we do not use any compression algorithms for the FM sketches to be able to tune the size of the CMFM sketch more accurately. In general, compression would reduce the overall size of the sketch to 30% of the uncompressed version, according to our measurements. In Figure 10(a) we plot the accuracy of the sketch as a function of the number of bitmaps used per FM sketch. We run the experiment using from 100 to 500 FM sketches (counters) per hash function, and 2 hash functions per sketch. A good trade-off between sketch size and estimation accuracy with small variance is using 40 bitmaps per FM and 100 FMs per hash. This is more clearly seen in Figure 10(b) where we plot the same results, this time showing the size of each sketch in KBytes. It is clear from the graphs that increasing the number of FMs per hash does not help improve accuracy. Nevertheless, it does have a positive impact on reducing variance (small error bars have been removed from the figure not to clutter the graph). Also, we would like to point out that the results for the FM sketches are better than the guarantees provided based on Theorem 6. Thus, the results of CMFM are better than the expected results based on the analysis in Section 4.

The next graphs show the estimation accuracy of the sketch as the link loss probability between child/parent sensors and the node loss probability increase. In this case we compute the estimation error with respect to the exact answer that would have been received at the sink, assuming zero losses. As expected, the estimation error increases proportionately as the link losses increase, since more and more client messages never reach any of their designated parent sensors (see Figure 11(a)). We observe similar results as the node loss probability increases (see Figure 11(b)), the only difference being that the error increases at a steeper rate since the failure of a single node results in complete loss of its data, unlike link losses where all links need to fail simultaneously for the data to be lost.



(a) average error as a function of link loss probability

(b) average error as a function of node loss probability

Fig. 11. Frequency estimation error as a function of link and node losses.

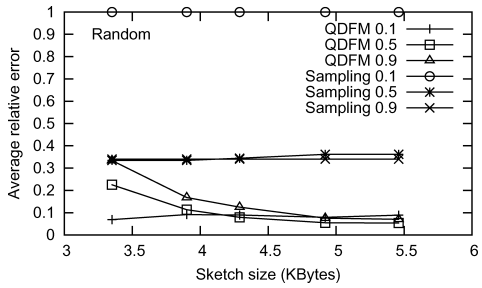


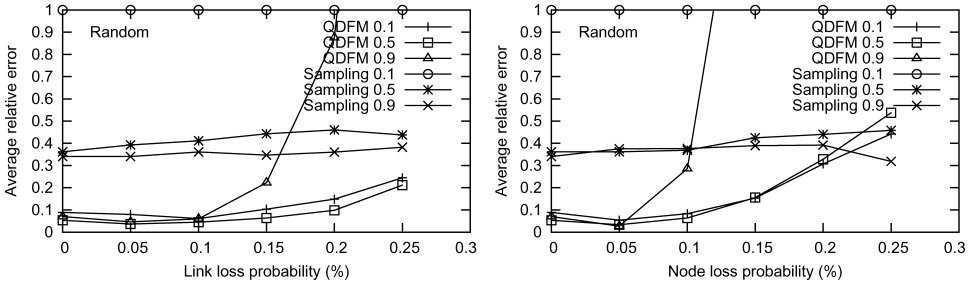
Fig. 12. Average error as a function of sketch size for uniformly distributed data.

6.3 Quantile Estimation

Finally, we evaluate the quantile estimation sketches. We use a 30×30 sensor grid, where each sensor generates 1000 values in [10, 100010] (we avoid small values here, since we are estimating quantiles, and small numbers will skew relative errors significantly even for small absolute errors). Here, we compute the estimation accuracy of the sketch by estimating the 10th percentile, the median, and the 90th percentile of all the values generated by the sensors in a single epoch, and averaging over all epochs (for a total of 30 epochs). We run two experiments, one with randomly distributed data, and one with a Zipf distribution, with parameter 1.2. We choose the compression factor k equal to 20. Experiments showed that this is a good compression ratio for our datasets. We compare our technique against the random sampling approach [Nath et al. 2004].

Figure 12 plots the estimation accuracy of the QDFM sketch and random sampling as a function of sketch size, for uniformly distributed values. Clearly, with a 5KByte sketch the estimation error of QDFM is below 10% for all quantiles. Random sampling yields 40% errors for the same sample size for the 50th and 90th percentile, while it completely fails to estimate values close to the minimum.

The accuracy of the sketches for increasing link and node loss probabilities is shown in Figures 13(a) and 13(b). Random sampling is unaffected by the losses at an almost constant, close to 40% error. QDFM is very resilient for up to 25%



(a) average error as a function of link loss probability

(b) average error as a function of node loss probability

Fig. 13. Quantile estimation error as a function of link and node losses for uniformly distributed data.

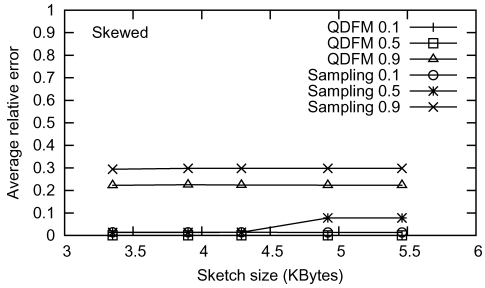


Fig. 14. Average error as a function of sketch size for skewed data distribution.

losses for the 10th and 50th percentile, but fails after 15% losses for the 90th percentile. Larger quantiles are harder to estimate using the QDFM due to the additive error introduced by the FM sketches as we aggregate nodes during quantile estimation (the more nodes we merge, the larger the error becomes).

Finally, we run experiments using skewed data distributions. We compare QDFM with random sampling in Figure 14. We can observe that both techniques can very accurately estimate the 10th and 50th percentile, since smaller values are the most frequent comprising the heavy tail of the distribution. Even here, though, QDFM is more accurate than sampling (0% versus 0.01% relative error). Both sketches have very large estimation error for the 90th percentile (up to 30% for sampling), which is expected since these are rare values in the data distribution. Nevertheless, the QDFM is still better than sampling. The techniques work similar to the uniform distribution case for link and node losses.

7. DISCUSSION

The sketches that we presented in this article can be used to provide approximate answers to several important aggregation queries in a sensor network system. It is shown, both analytically and experimentally, that the distinct and summation FM sketches provide a good trade-off between accuracy and communication cost on a sensor network that exhibits low or medium link

and node losses. In particular, for loss rates between 1% to 30% the aforementioned sketches provide a good level of accuracy on the estimation of SUM and COUNT queries. However, the accuracy of these methods decreases for higher loss rates (although at a slower rate than other methods). For sensor networks with very low loss rates and stable topologies, TAG-based approaches are the best alternative. In cases for which a single network exhibits areas with high losses and other areas with limited or no losses, these two methods can be combined efficiently [Manjhi et al. 2005]. For the estimation of frequent values in a lossy sensor network, the CMFM sketch seems to be the best alternative. In addition, it provides strong probabilistic guarantees that are validated by our experimental evaluation. However, if the loss rate surpasses 25%, the accuracy of the estimator starts to drop. Therefore, we again recommend the use of CMFM for the case of low to medium link and node failures. Finally, for estimating quantiles in a sensor network with no losses, the q-digest provides a very good approximate solution. For the case of losses, the QDFM sketch can be used, but without analytic guarantees on the quality of the results. Indeed, in our experimental study we found cases where the QDFM has very high error, even under medium loss rates. Devising a duplicate-insensitive variant of the QDFM sketch with comparable analytic and creating performance guarantees to the basic sketch is an important open problem.

8. CONCLUSIONS AND FUTURE WORK

We have presented new methods for approximately computing duplicate-sensitive aggregates across distributed datasets. Our immediate motivation comes from sensor networks, where energy consumption is a primary concern, faults occur frequently, and exact answers are not required or expected. An elegant building block which enables our techniques is comprised of the duplicate-insensitive sketches of Flajolet and Martin, which give us considerable freedom in our choices of how best to route data and where to compute partial aggregates. In particular, use of this duplicate-insensitive data structure allowed us to make use of dispersity routing methods to provide fault tolerance that would be inappropriate otherwise. Of course, our approach does not come for free. Since we duplicate results, the power consumption of our technique is higher than single-path techniques. It would be interesting to investigate the trade-off between energy consumption and robustness to node and link failures using appropriate network models [Chakrabarti et al. 2007]. Furthermore, certain applications may require exact results or the original readings of the sensor and not aggregate results. In that case, any approximate in-network aggregation method cannot be applied.

The implications of these results reach beyond sensor networks to other unreliable systems with distributed datasets over which best-effort aggregate queries are posed. Examples include estimating the number of subscribers participating in a multicast session, or counting the number of peers storing a copy of a given file in a peer-to-peer network. In these settings, nodes are less resource-constrained than in sensor networks, but the problems are still difficult due to packet loss and frequent node arrivals and departures.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions and S. Madden for providing an early version of the TAG simulator.

REFERENCES

ADURI, P. AND TIRTHAPURA, S. 2005. Range efficient computation off 0 over massive data streams. In *Proceedings of the International Conference on Data Engineering (ICDE'05)*, 32–43.

ALON, N., MATIAS, Y., AND SZEGEDY, M. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'96)*, 20–29.

BAR-YOSSEF, Z., JAYRAM, T. S., KUMAR, R., SIVAKUMAR, D., AND TREVISAN, L. 2002. Counting distinct elements in a data stream. In *Proceedings of the International Workshop on Randomization and Approximation Techniques (RANDOM'02),* 1–10.

BAWA, M., GARCIA-MOLINA, H., GIONIS, A., AND MOTWANI, R. 2003. Estimating aggregates on a peer-to-peer network. Tech. rep., Stanford University.

CHAKRABARTI, D., LESKOVEC, J., FALOUTSOS, C., MADDEN, S., GUESTRIN, C., AND FALOUTSOS, M. 2007. Information survival threshold in sensor and p2p networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (InfoCom'07),* 1316–1324.

CONSIDINE, J., LI, F., KOLLIOS, G., AND BYERS, J. 2004. Approximate aggregation techniques for sensor databases. In *Proceedings of the International Conference on Data Engineering (ICDE'04),* 449–461.

CORMODE, G., DATAR, M., INDYK, P., AND MUTHUKRISHNAN, S. 2003. Comparing data streams using hamming norms (how to zero in). *IEEE Trans. Knowl. Data Eng.* 15, 3, 529–540.

CORMODE, G. AND MUTHUKRISHNAN, S. 2005a. An improved data stream summary: The count-min sketch and its applications. *J. Algor. 55*, 1, 58–75.

CORMODE, G. AND MUTHUKRISHNAN, S. 2005b. Space efficient mining of multigraph streams. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'05),* 271–282.

DELIGIANNAKIS, A., KOTIDIS, Y., AND ROUSSOPOULOS, N. 2004a. Compressing historical information in sensor networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 527–538.

DELIGIANNAKIS, A., KOTIDIS, Y., AND ROUSSOPOULOS, N. 2004b. Hierarchical in-network data aggregation with quality guarantees. In *Proceedings of the International Conference on Extending Database Technology (EDBT'04),* 658–675.

DELIGIANNAKIS, A., KOTIDIS, Y., AND ROUSSOPOULOS, N. 2007. Bandwidth-Constrained queries in sensor networks. *The VLDB J. 16*, 3.

DESHPANDE, A., GUESTRIN, C., MADDEN, S., HELLERSTEIN, J., AND HONG, W. 2004. Model-Driven data acquisition in sensor networks. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'04),* 588–599.

DESHPANDE, A. AND MADDEN, S. 2006. Mauvedb: Supporting model-based user views in database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 73–84.

DEVROYE, L. 1986. *Non-Uniform Random Variate Generation*. Springer.

DURAND, M. AND FLAJOLET, P. 2003. Loglog counting of large cardinalities. In *Proceedings of the European Symposium on Algorithms (ESA'03)*, 605–617.

FLAJOLET, P. 1990. On adaptive sampling. *COMPUTG: Comput. 43*.

FLAJOLET, P. AND MARTIN, G. N. 1985. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci. 31*, 2, 182–209.

FLOYD, R. W. AND RIVEST, R. L. 1975. Expected time bounds for selection. *Commun. ACM 18*, 3, 165–172.

GANESAN, D., GOVINDAN, R., SHENKER, S., AND ESTRIN, D. 2001. Highly-Resilient, energy-efficient multipath routing in wireless sensor networks. *ACM Mobile Comput. Commun. Rev. 5*, 4, 11–25.

GANGULY, S., GAROFALAKIS, M., AND RASTOGI, R. 2003. Processing set expressions over continuous update streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 265–276.

GAROFALAKIS, M., HELLERSTEIN, J. M., AND MANIATIS, P. 2007. Proof sketches: Verifiable in- network aggregation. In *Proceedings of the International Conference on Data Engineering (ICDE'07),* 996–1005.

GIBBONS, P. AND TIRTHAPURA, S. 2001. Estimating simple functions on the union of data streams. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, 281–291.

GRAY, J., CHAUDHURI, S., BOSWORTH, A., LAYMAN, A., REICHART, D., VENKATRAO, M., PELLOW, F., AND PIRAHESH, H. 1997. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining Knowl. Discov. 1*, 1, 29–53.

GREENWALD, M. AND KHANNA, S. 2004. Power-Conserving computation of order-statistics over sensor networks. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS'04),* 275–285.

HORTON, M., CULLER, D., PISTER, K., HILL, J., SZEWCZYK, R., AND WOO, A. 2002. Mica, the commercialization of microsensor motes. *IEEE Sensors J. 19*, 4, 40–48.

Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. 2003. Directed diffusion for wireless sensor networking. *IEEE/ACM Trans. Netw. 11*, 1, 2–16.

KAMRA, A., MISRA, V., AND RUBENSTEIN, D. 2007. Counttorrent: Ubiquitous access to query aggregates in dynamic and mobile sensor networks. In *Proceedings of the SIGOPS International Conference on Embedded Networked Sensor Systems (SenSys'07),* 43–57.

KEMPE, D., DOBRA, A., AND GEHRKE, J. 2003. Gossip-Based computation of aggregate information. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'03),* 482.

KOTIDIS, Y. 2005. Snapshot queries: Towards data-centric sensor networks. In *Proceedings of the International Conference on Data Engineering (ICDE'05),* 131–142.

MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2002. Tag: A tiny aggregation service for ad-hoc sensor networks. *ACM SIGOPS Oper. Syst. Rev. 36*, *SI*, 131–146.

MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2003. The design of an acquisitional query processor for sensor networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 491–502.

MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2005. Tinydb: An acquisitional query processing system for sensor networks. *ACM Trans. Database Syst. 30*, 1, 122–173.

MANJHI, A., NATH, S., AND GIBBONS, P. B. 2005. Tributaries and deltas: Efficient and robust aggregation in sensor network streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 287–298.

MUNRO, J. I. AND PATERSON, M. 1980. Selection and sorting with limited storage. *Theor. Comput. Sci. 12*, 315–323.

NATH, S., GIBBONS, P. B., SESHAN, S., AND ANDERSON, Z. R. 2004. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the International Conference on Embedded Networked Sensor Systems (SenSys'04),* 250–262.

PALMER, C. R., GIBBONS, P. B., AND FALOUTSOS, C. 2002. ANF: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 81–90.

PAPOULIS, A. 1965. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York.

PRZYDATEK, B., SONG, D., AND PERRIG, A. 2003. SIA: Secure information aggregation in sensor networks. In *Proceedings of the SIGOPS International Conference on Embedded Networked Sensor Systems (SenSys'03),* 255–265.

SHARAF, A., BEAVER, J., LABRINIDIS, A., AND CHRYSANTHIS, K. 2004. Balancing energy efficiency and quality of aggregate data in sensor networks. *The VLDB J. 13*, 4, 384–403.

SHRIVASTAVA, N., BURAGOHAIN, C., AGRAWAL, D., AND SURI, S. 2004. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the SIGOPS International Conference on Embedded Networked Sensor Systems (SenSys'04),* 239–249.

STOEV, S., HADJIELEFTHERIOU, M., KOLLIOS, G., AND TAQQU, M. S. 2007. Norm, point, and distance estimation over multiple signals using max-stable distributions. In *Proceedings of the International Conference on Data Engineering (ICDE'07)*, 1006–1015.

U.S. CENSUS BUREAU. 2009. TIGER/Line datasets. http://www.census.gov/geo/www/tiger/.

WALKER, A. 1977. An efficient method for generating discrete random variables with general distributions. *ACM Trans. Math. Softw. 3*, 3, 253–256.

YAO, Y. AND GEHRKE, J. 2002. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec. 31*, 3, 9–18.

YAO, Y. AND GEHRKE, J. 2003. Query processing in sensor networks. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR'03)*.

YE, F., ZHONG, G., LU, S., AND ZHANG, L. 2005. Gradient broadcast: A robust data delivery protocol for large scale sensor networks. *ACM Wireless Netw.11*, 3, 285–298.

ZHAO, J., GOVINDAN, R., AND ESTRIN, D. 2003. Computing aggregates for monitoring wireless sensor networks. In *Proceedings of the IEEE International Workshop on Sensor Network Protocols and Applications (SNPA'03)*,139–148.

Received February 2007; revised June 2008; accepted August 2008